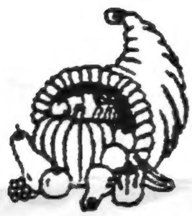
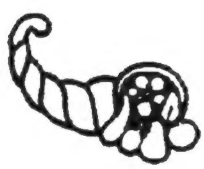
# LISTing Newsletter

**Newsletter of the Long Island Sinclair/Timex Users Group**

| NOVEMBER 1994 | | | | | | |
|---|---|---|---|---|---|---|
| SU | MO | TU | WE | TH | FR | SA |
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | | | |
| LIST MEETING 13TH | | | | | | |





*Listing Policy*

*Annual Dues $16.00*

One "sample" copy sent upon receipt of Business size SASE. Copies provided on EXCHANGE BASIS with other bona fide user groups. LISTing is published monthly except July and August by LIST (Long Island Sinclair Timex) Group, a not for profit user group.

We are always looking for articles, programs, reviews etc. to keep our members informed and entertained. You maintain full credit and copyright.

Portions if this publication may be reproduced, without written consent. Please give credit to LIST when reprinting articles.

LIST disclaims any responsibility for any damage you may do to your computer as a result of reading any articles in LISTing.

## LIST OFFICERS

| | |
|---|---|
| Pres. | Harvey Rait |
| V. P. | Bob Gilder |
| Tres. | Robert Malloy |
| Cor. Sec. | John Pazmino |
| Editor | Fred Stern |
| Libr. | Tom Skapinski |

Please send inquiries to:
LIST
Mr. Harvey Rait
5 Peri Lane,
Valley Stream, N. Y. 11581

Please send submissions to:

Mr. Frederic Stern
P. O. Box 264,
Holbrook, N. Y. 11741

Coming Events:

LIST Meeting October 16, 1994

- Special Notice -

The next meeting will be held at:
The Ice Cream Dispensary
(Harveys Store)
334 Dogwood Avenue,
Franklyn Square, N. Y.
Tel: 516-486-1090

Directions: Southern State Parkway to exit 17, North (Hempstead Ave.) Go to first traffic light, Left turn on to Cornwall, Next traffic light, bear right on to Dogwood Avenue, Go one mile to the Ice Cream Dispensary, in a small shopping center on the left side of the road.


## A Final Word

My Name is Fred Stern and I am the Editor of this edition of Listing.

Meeting Minutes

Reported by Fred Stern

The meeting was called to order by Harvey at 2 PM.

In the mail, we received one correspondence. Fred will answer.

We only received two newsletters from the exchange.

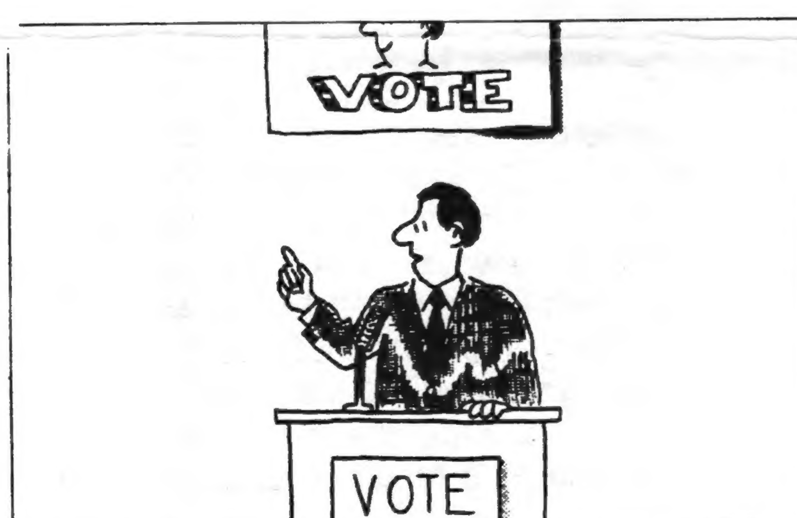
We were sadden to receive our last issue of Sinc-Linc. It was one of best Timex newsletter from Canada. It will be missed.

We had a roundtable discussion about the John Oliger EPROM Burner and how to use it with the TS 1000.

January is renewal time, and it is coming up soon. Don't forget to renew your membership to LIST.

Our next meeting will be on Nov. 13, 1944 at The Ice Cream Dispensary.

HAVE A VERY HAPPY THANKSGIVING TO ALL LIST MEMBERS AND THEIR FAMILIES.




*"I'm the only political candidate to take a stand on computer operating systems."*

# QL CORNER

I received a call from Bob Dyl stating that he has resigned as the USA, Eastern QUANTA sub-librarian. The work load for IQLR is extremely demanding and now that IQLR has it's own software library. Bob felt that it would be considered as a conflict of interest if he remained in this position.

Bob requested that I take over this position and that he had recommended me to QUANTA. QUANTA has now endorsed this change for me to be YOUR sub-librarian. When library updates are received, they will appear in IQLR and LIST's monthly QL Corner and in UPDATES. I can accommodate QUANTA members with almost any disk drive size and density: 3.5 inch disks 720K, 1.4M and 3,2M, and 5.25 inch disks in either 360K DD and 720K QD.

Every once in a while one of members has a QL with an intermittent problem. Most times, I just lever up all of the ICs in sockets, press them down and the QL is up and running. However, this fix is only temporary since the QL's IC leaf spring sockets are of poor quality. The best solution for this problem is to insert machine pin IC sockets into the original QL IC sockets. Normally 40 pin machine pin sockets sell for $1.00 - $1.59 each. There is a source for high quality machine pin sockets at a low price, B. G. MICRO, INC. The cost for 40 pin machine pin sockets are 3 for $1.00. In fact, they sell 8, 14, 16, 18, 20, 24, 28 pin sockets for the same price, 3/$1.00. If you require two 28 pin sockets for the roms, you should purchase 40 pin sockets and cut them down to size.

For those members who like to prototype let's say, a rom switch board, these B. G. Micro's machine pin sockets have extra long pins and the socket base is also half the thickness of other sockets which provides the user with high quality low profile sockets at a third of the normal price. The only catch is that they require a $10.00 minimum. Perhaps several QL users could get together for a bulk purchase.

Digital Precision has announced in their ad in IQLR, that they are offering a software deal to QL users; approximately 50 programs for '149.00 including shipping charges. Each program has a manual supplied on to disk. This saves Digital Precision money on postage. And if you aren't interested in some of the programs, you need not print out the manuals. This offer will only be in effect for two weeks. I had placed my order as soon as I had received the latest issue of IQLR. DP accepts credit cards. I am sure that all files are Zipped, again, to save on postage......we will see.

This past month I have been busy prototyping several ROM switch's for my PC cased QLs. I duplicated the Minerva style pc board for each ROM adding a 10K resistor at Eprom pin #22 for +5V DC. The trace which joins the outer pin #22 of the Eprom board to pin #8 of the HCT10 has been opened. Three leads are soldered from each of these points to a DPDT switch. See the following diagram:

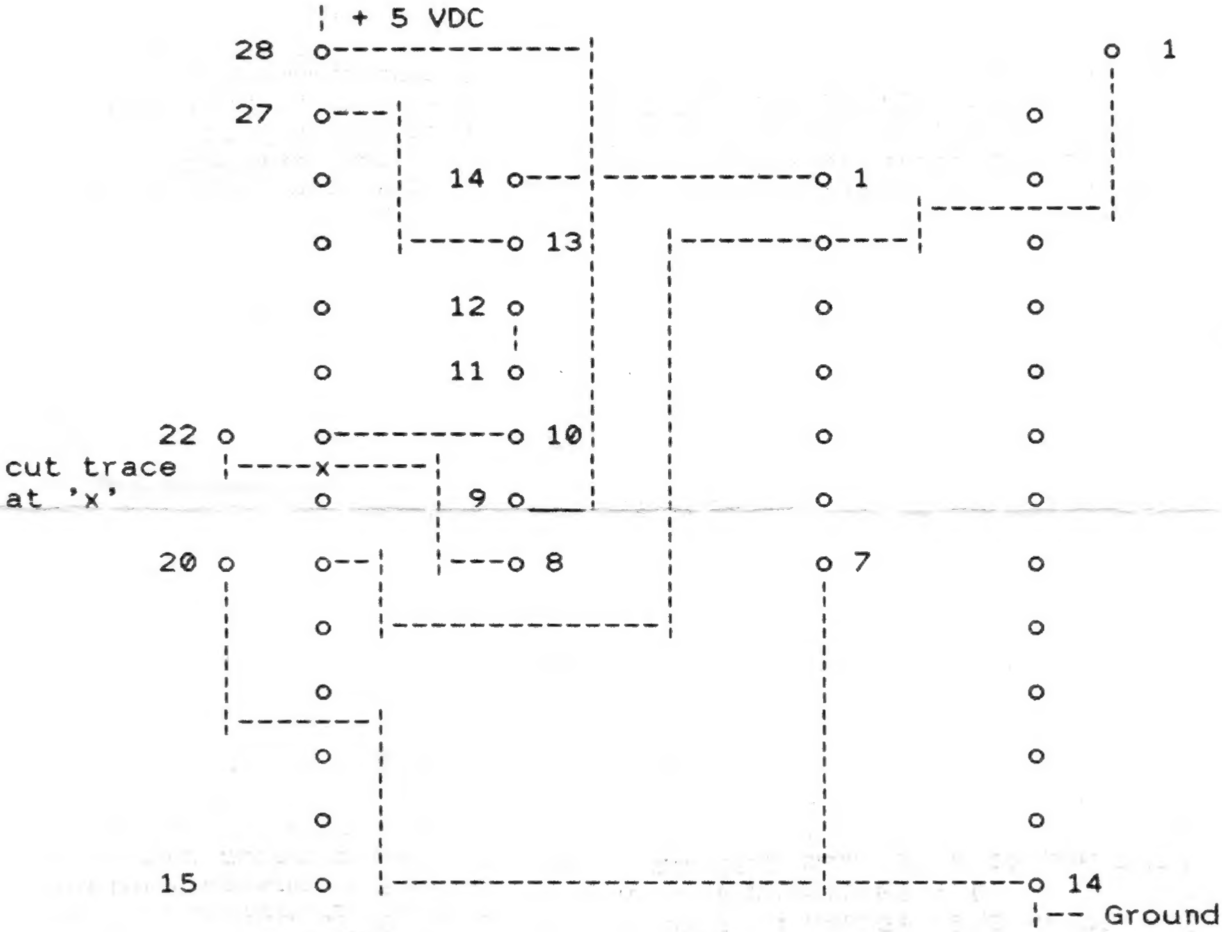
```
MINERVA ROM          SPDT Switch       JSU ROM

+ 5VDC-/\/\/\--------o           o---------- #8, 74HCT10
     10K ohm


#22, Eprom-----------o           o---------- #22, Eprom


 #8, 74HCT10---------o           o--/\/\/\------ +5VDC
                                 10K ohm
```

MINERVA EPROM BOARD
(BOTTOM VIEW OF BOARD)




Eprom 27C156

TTL 74HCT10

DISABLE EPROM

For a second Eprom ROM board, the switch is wired in the reverse order allowing either ROM (a MINERVA and a JSU) to be switched.

Both Eprom boards are installed on a small Mother board containing two 40-pin machine pin sockets. Pins 2 thru 28 of each socket are wired in parallel. In other words, pin 2 from socket 1 is connected to pin 2 of socket 2 ... and so on. When time allows I will produce photographic negatives for the Mother board and Eprom board and publish them in the QL corner. If one has access to a good photo copier then the PC boards can be duplicated on to a mylar film, such as TEC-200 copying film. The film is then ironed onto PC board material and etched for a perfect duplicate of the PC board.

I have ordered the new SMSQ\E QL operating system from Jochen Merz. I should receive the software package approximately at the beginning of December. If I get the hang of it right away, I will review it in the December issue of LISTing.

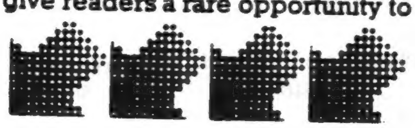
Se you in December.......Bob Gilder

# Give your power pills more punch

**Tony Rickwood demonstrates how much faster a game will work when Basic is replaced by machine code in key areas.**

IF YOU have followed this series on machine code so far, you will by now, be raring to go with some practical games applications. Most commercial games are 100% machine coded for speed and animation quality, but it would be much too ambitious for us to present and explain what might be an enormous assembler listing in these few pages.

A much more practical approach is to see how a traditional maze type game, written in Basic, can be made much more fun to play by replacing some of the routine basic programming with machine code. Admittedly, maze games are a bit old hat now, but the game I will be presenting here has been specially written to give readers a rare opportunity to



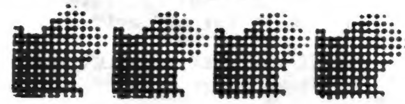
see, at a glance, just how much better than Basic m/c really is.

At the touch of a key, you will be able to get your Spectrum to throw out a large chunk of Basic and call in a m/c routine instead. See for yourself how much more energy a diet of power pills and m/c will give Pacman and the four ghosts.

First, enter and run Program one which POKEs 199 bytes of machine code into memory, starting at location 64000 (the game is designed for the 48K Spectrum). As a wrong number may cause the computer to crash, a check is made to make sure data is entered correctly. When you have got the "data entry o.k." message, SAVE program 1 to tape.

Now enter Program two. When entering a long Basic program, I find that the best plan is to start by entering and testing the part which sets up the User Defined Graphics. This is contained in lines 6000 onwards. Now RUN this part (do not worry about the "RETURN WITHOUT GOSUB" message) and go into graphics mode (Caps/9). Pressing any letter from a to u will demonstrate that all 21 UDGs have been used. Graphic letters a to n are for the maze walls, o to u are for Pacman (moving in four directions), ghosts, power pills, and dots.

The next part to enter and check is the subroutine which sets up the screen (in this case, by printing the maze). Lines 5000-5040 do this. With the graphics already set up in memory, you will see the maze emerge as you enter the two PRINT statements, which will help you to correct any mistakes as they are made. Now type RUN 6000 to make sure that the maze is printed complete with dots and power pills. You should have everything in white on a black background.



As the way the program works depends on different objects having different colours, these are best put into the PRINT statements using colour control characters. The colours you MUST

```
  10 REM Machine Code Setup for
Ghostchase @ Tony Rickwood
  20 CLEAR 63999: LET s=0: FOR i
=64000 TO 64198: READ n: POKE i,
n: LET s=s+n: NEXT i
  30 READ sum: IF s <> sum THEN
 PRINT "error in data entry"'"ch
eck line 40": STOP
  40 DATA 97,107,98,108,99,109,1
00,110,101,111,102,112,103,113,1
04,114,120,121,97,98,99,100,101,
102,103,104,0,221,33,0,250,6,8,2
05,153,250,126,87,221,35,205,153
,250,114,221,35,16,241,205,153,2
50,70,221,35,205,153,250,78,221,
35,221,126,0,254,0,200,205,153,2
50,94,221,35,205,153,250,86,221,
43,120,147,40,28,56,11,28,205,16
9,250,254
  50 DATA 6,32,14,29,24,15,29,20
5,169,250,254,6,32,3,28,24,4,205
,153,250,115,221,35,121,146,40,2
8,56,11,20,205,169,250,254,6,32,
14,21,24,15,21,205,169,250,254,6
,32,3,20,24,4,205,153,250,114,22
1,35,205,169,250,54,6,24,163,197
,42,75,92,1,200,0,221,126,0,237,
177,35,35,193,201,213,197,33,0,8
8,75,90,22,0,6,5,203,35,203,18,1
6,250,123,137,48,1,20,95,25,126,
193,209,201,209,201,22690
  60 PRINT "Data entry o.k"'"m/c
 now ready for running": STOP
  70 SAVE "GCODE" CODE 64000,199
```

*Program two*

```
   10 REM GHOSTCHASE @ Tony Rickw
ood
   20 GO SUB 100
   25 LET s=0: LET v=5
   30 GO SUB 6000
   45 GO SUB 5000
   50 INPUT "Basic or m/c(b/m)?";
g$
   51 IF g$ <> "b" AND g$ <> "m"
THEN  GO TO 50
   60 GO SUB 1000
  100 REM Initialize control vari
able
  110 LET a=3: LET b=1: LET c=27:
 LET d=1: LET e=3: LET f=19: LET
 g=27: LET h=19: LET x=15: LET y
=10
  120 LET k=a: LET l=b: LET m=c:
LET n=d: LET o=e: LET p=f: LET q
=g: LET r=h
  130 RETURN
 1000 REM Main Control Routine
 1005 LET di=1: LET s=s+1
 1010 LET xx=x: LET yy=y: PRINT
AT yy,xx;" "
 1020 PRINT  INK 1; OVER 1; AT b,
a;"T"; INK 2; AT d,c;"T"; INK 3;
 AT f,e;"T"; INK 4; AT h,g;"T":
BEEP .005,45
 1025 IF  ATTR (y,x) <= 4 THEN  G
O TO 3000
 1030 LET x=x-( INKEY$ ="5")+( IN
KEY$ ="8"): LET y=y+( INKEY$ ="6
")-( INKEY$ ="7")
 1035 LET di=di*(xx=x)*(yy=y)+(x>
xx)+2*(x<xx)+3*(y<yy)+4*(y>yy)
 1036 IF x=1 THEN  LET x=28
 1037 IF x=29 THEN  LET x=2
 1040 IF  ATTR (y,x)=6 THEN  LET
x=xx: LET y=yy: GO TO 1070
 1055 IF  ATTR (y,x)=5 THEN  LET
s=s+10: FOR i=1 TO 10: BEEP .005
,20: BEEP .005,30: NEXT i: GO TO
 1070
 1060 IF  POINT (3+x*8,171-8*y)=1
 THEN  LET s=s+1: BEEP .005,30
 1070 PRINT  AT 21,8;s: PRINT  AT
 y,x;"OPQR"(di)
 1080 IF  ATTR (y,x) <= 4 THEN  G
O TO 3000
 1082 IF g$="m" THEN  RANDOMIZE
USR 64027
 1090 IF g$="b" THEN  GO SUB 4000
 1100 PRINT  OVER 1; AT l,k;"T";
AT n,m;"T"; AT p,o;"T"; AT r,q;"
T"
 1110 GO TO 1010
 2000 REM Screen Cleared
 2010 GO SUB 100
 2020 GO SUB 5000
 2030 GO SUB 1000
 2040 RETURN
 3000 REM Pacman eaten
 3010 LET v=v-1: PRINT  AT 21,28;
v
 3020 FOR i=1 TO 20: BEEP .01,20:
 BEEP .01,30: NEXT i
 3030 IF v=0 THEN  GO TO 3040
 3035 PRINT  INK 7; OVER 1; AT b,
a;"T";; AT d,c;"T"; AT f,e;"T";
AT h,g;"T"
 3036 GO SUB 100
 3037 GO SUB 1000
 3040 CLS : PRINT  AT 10,0;"YOU S
CORED ";s; AT 11,0;"ANOTHER GAME
?(y/n)"
 3050 IF  INKEY$ ="" THEN  GO TO
3050
 3060 IF  INKEY$ ="n" THEN  STOP
 3070 RUN
 4000 REM Ghost Control
 4010 LET k=a: LET l=b: LET m=c:
LET n=d: LET o=e: LET p=f: LET q
=g: LET r=h
 4020 LET a=a+(x>a)-(x<a): IF  AT
TR (b,a)=6 THEN  LET a=k
 4030 LET b=b+(y>b)-(y<b): IF  AT
TR (b,a)=6 THEN  LET b=l
 4040 IF a <> k OR b <> l THEN  P
RINT  OVER 1; INK 6; AT b,a;" "
 4050 LET c=c+(x>c)-(x<c): IF  AT
TR (d,c)=6 THEN  LET c=m
 4060 LET d=d+(y>d)-(y<d): IF  AT
TR (d,c)=6 THEN  LET d=n
 4070 IF c <> m OR d <> n THEN  P
RINT  OVER 1; INK 6; AT d,c;" "
 4080 LET e=e+(x>e)-(x<e): IF  AT
TR (f,e)=6 THEN  LET e=o
 4090 LET f=f+(y>f)-(y<f): IF  AT
TR (f,e)=6 THEN  LET f=p
 4100 IF e <> o OR f <> p THEN  P
RINT  OVER 1; INK 6; AT f,e;" "
 4110 LET g=g+(x>g)-(x<g): IF  AT
TR (h,g)=6 THEN  LET g=q
 4120 LET h=h+(y>h)-(y<h): IF  AT
TR (h,g)=6 THEN  LET h=r
 4140 RETURN
 5000 REM Print Maze
 5005 PAPER 0: BORDER 0: INK 7: C
LS
 5010 PRINT "  CAAAAAAAAAAAAANAAAA
AAAAAAAAD"'"  BSSSSSSSSSSSSSBSSSS
SSSSSSSSB"'"  BSCADSCANAAISSSJAA
NADSCADSB"'"  BSB BSB BSSSSSSSSS
B BSB BSB"'"  BSEAFSEAFSJAANAAIS
EAFSEAFSB"'"  BSSSSSSSSSSSSSBSSSS
SSSSSSSSB"'"  BSJAISJANAAISSSJAA
NAISJAISB"'"  BUSSSSSSBSSSSSSSSS
BSSSSSSUB"'"  EAAAAADSBSCAISJADS
BSCAAAAAF"'"  AAAAAAFSGSBSBSSSBS
BSEAAAAAA"'"  SSSSSSSSSSBSSSSSBS
BSBBSSSSS "
 5020 PRINT ;"  AAAAAADBHSBSSSSSB
SHSCAAAAAA"'"  CAAAAAFSGSEAANAAF
SGSEAAAAAD"'"  BUSSSSSSSSSSSBSSS
SSSSSSSSUB"'"  BSJADSJAAAAISGSJA
AAAISCAISB"'"  BSSSBSSSSSSSSSSSS
BSSSSBSSSB"'"  KAISEAISHSJAANAAI
SHSJAFSJAL"'"  BSSSSSSSBSSSSBSSS
SBSSSSSSSB"'"  BSJAAAAAMAAISGSJA
AMAAAAAISB"'"  BSSSSSSSSSSSSSSSS
SSSSSSSSSB"'"  EAAAAAAAAAAAAAAAA
AAAAAAAAF"
 5030 PRINT  AT 21,2;"SCORE"; AT
21,22;"LIVES ";v
 5040 RETURN
 6000 REM UDG Setup
 6010 FOR i= USR "a" TO  USR "u"+
7: READ j: POKE i,j: NEXT i
 6020 DATA 0,255,j,j,j,j,j,0
 6030 DATA 126,j,j,j,j,j,j,j
 6040 DATA 0,31,63,127,j,j,j,126
 6050 DATA 0,248,252,254,j,j,j,12
6
 6060 DATA 126,127,j,j,j,63,31,0
 6070 DATA 126,254,j,j,j,252,248,
0
 6080 DATA 126,j,j,j,j,60,24,0
 6090 DATA 0,24,60,126,j,j,j,j
 6100 DATA 0,248,252,254,j,252,24
8,0
 6110 DATA 0,31,63,127,j,63,31,0
 6120 DATA 126,127,j,j,j,j,j,126
 6130 DATA 126,254,j,j,j,j,j,126
 6140 DATA 126,255,j,j,j,j,j,0
 6150 DATA 0,255,j,j,j,j,j,126
 6160 DATA 62,127,248,240,j,248,1
27,62
 6170 DATA 124,255,31,15,j,31,254
,124
 6180 DATA 66,195,j,231,255,j,126
,60
 6190 DATA 60,126,255,j,231,195,j
,66
 6200 DATA 0,0,0,24,24,0,0,0
 6205 DATA 60,126,219,255,255,255
,165,165
 6210 DATA 0,0,60,126,j,60,0,0
 6220 RETURN
```

use are: WALLS = yellow, DOTS = white, POWER PILLS = cyan.

The way to put these codes into the PRINT statements is described in Chapter 16 of the Spectrum manual, but to get you started, first bring down line 5010 with edit. Run the cursor to just past the quotes and change to extended mode (caps shift/symbol shift). Now hold the Capitals key and press key 6. You will see that everything that follows is now printed in yellow ink. Now run the cursor right by one space (to just before the first dot), change cursor to Extended again, and press Caps/7 to change back to white ink. Continue throughout until everything is the correct colour.

IMPORTANT NOTE: The game depends on having these colours right. It also depends on the two exits from the maze being sealed to the ghosts (but not to Pacman), to allow Pacman to shake them off. This is done by putting an invisible wall at the two exits by printing the blank character positions to the left and right of the middle row of dots in yellow ink with the appropriate colour code in line 5010. You will see how this works when I explain ghost control.

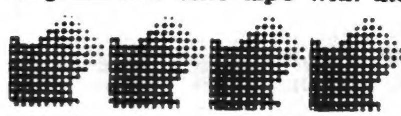
Now you can enter the rest of the program and test it as a whole, although it would be a good idea to do an intermediate SAVE of your work so far.


When RUNning the whole program, it will begin by asking whether you want Basic or M/C? This gives you the option of letting your Spectrum control the ghosts, using EITHER the Basic in lines 4000-4140 OR by using the machine code set up by Program one. They do exactly the same

job, though with a remarkable difference in speed. To satisfy yourself that these are directly interchangeable, look at lines 1082/1090 which call the appropriate routine according to the response to the only prompt used. Make sure you give the program a thorough testing in


each mode, although I guarantee you will not be using the Basic option for very long!

If you want to keep **Ghostchase** in your library of games, it would be a good idea to put the m/c itself (not Program one, which only sets it up) onto tape immediately after your dump of Program two. First, get a new dump of Program two onto tape with the



following line added: 15 LOAD "GCODE" CODE. If you want auto start, then dump by typing SAVE "ghost" LINE 1. Assuming that the m/c is in memory, dump it onto tape using SAVE "GCODE" CODE 64000,199. LOADing "ghost" will now automatically LOAD the m/c and RUN.

A few words about playing the game, as I have not included instructions in the program. Pacman control is by cursor keys. As usual, eat as many dots and power pills (10 point bonus) without getting caught, though you do have five lives. A screenful (300 points) earns you another screenful.

Obviously, you want to know how the machine code works, because, unlike routines I have presented so far, this one is not


portable. Special effects m/c routines seen previously can be used by any Basic program. They are portable in the sense that they do not depend on, or affect anything else, and are also relocatable to use any available memory space. The routine used here obviously does affect, and is affected by, the environment in which it operates. So, if you want to supercharge a chase routine of your own (not necessarily of the maze variety), you really must be able to understand it and adapt it to your own program.

The key to understanding the machine code is the Basic routine in lines 4000-4140. The principle is always to get the computer to calculate intelligently the new coordinates of the chaser(s) in order to place it nearer the character being chased. There are many levels of artificial intelligence which could be used (to prevent the chaser from staying blocked for example). Here, though, it is simply a matter of getting the four ghosts nearer without going through walls. Line 4010 starts by storing all the old coordinates as figure one:

Fig. one

| Ghost | New/Current Coords | | Coords | |
|---|---|---|---|---|
| | x | y | x | y |
| Blue | a | b | k | l |
| Red | c | d | m | n |
| Magenta | e | f | o | p |
| Green | g | h | q | r |

Pacman coordinates are called x,y. Before calculating new positions, current positions must be stored. This is to be able to erase old ghosts and for re-setting positions if a move is invalid. Lines

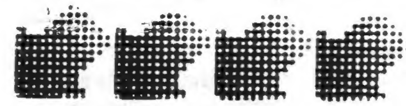
4020 and 4030 deal with x and y coordinates for the blue ghost (a,b). Using x>a and x<a for pacman to right or left of the blue ghost, a is increased or decreased by one.

Similarly for b in the vertical sense (note: x=a or y=b means no change). Both a and b must be tested separately to allow the ghost to move horizontally, vertically or diagonally toward Pacman. The only test for a valid move is that it does not collide with a wall.

No off screen tests are necessary because the ghosts can never get out of the maze. So all we need to test is ATTR(b,a)=6 to tell the computer that there is a (yellow) wall at the new position


calculated for the blue ghost. If true, the position reverts back to k and/or l. Other ghosts are handled in the same way.

What about collision with Pacman and other ghosts? The "Pac-



man eaten" condition is picked up in the main control routine at line 1080. Collision with another ghost is a bit more subtle. We cannot test ATTR(b,a)<=4, for example, to see if there is another ghost already there, for the simple reason that no new ghosts can be printed until ALL new positions have been calculated. Line 4030 solves the problem for the blue ghost (4070 and 4100 for magenta and green). In effect, this line is saying that, if the blue ghost has been moved (a<>k) or (b<>l) then temporarily PRINT a piece of invisible wall OVER the new position (to preserve the dot) to prevent it from being occupied by another ghost. This is only temporary while computer control is inside the ghost control subroutine.

Soon after control is returned to the main program (starting at line 1000), the ghosts are printed and so overwrite the invisible wall. Obviously, no such device is required for the green ghost as this is the last to be moved.

Do try to understand thoroughly how the subroutine starting at line 4000 works. You will then have a much better chance of understanding the machine coded version which I will be explaining in the next article.

# Machine code refreshes parts Basic cannot reach

**Last month we showed how machine code can add speed to a Basic game. Tony Rickwood gives more details.**

LAST MONTH, I showed how a piece of machine code programming could dramatically improve the speed at which four ghosts could chase **Pacman.** Much of the programming is still in Basic. The purpose of the m/c is to replace a Basic subroutine by which the computer gives chase.

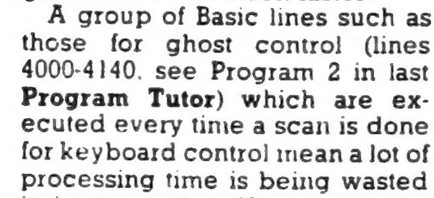
Why does the m/c make the ghosts move so much faster?

A group of Basic lines such as those for ghost control (lines 4000-4140, see Program 2 in last **Program Tutor**) which are executed every time a scan is done for keyboard control mean a lot of processing time is being wasted in interpretation. No wonder the four ghosts look so tired! The machine code gives time a shot in the arm by allowing them to escape the bonds of Sinclair's interpreter!

To supercharge your own maze games, you need to understand how my routine works to be able to adapt it. I will assume you have Programs one and two from Part One on cassette, with the m/c also dumped (from Program One) and called "GCODE".

**What GCODE does**

The bytes of m/c are assem-



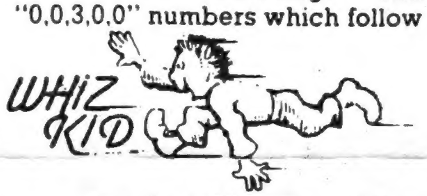

bled from the listing in Figure 1 using a commercial assembler program. If you really want to start writing your own code or adapting other people's, a good assembler is essential. So forget the games for a while and save your money for a useful piece of software instead!

Like the Basic subroutine it replaces, the assembler listed in Figure 1 works on the VARS (variables) area of memory to access and update the ghost coordinates. Of course, the Basic conceals how this is done. To understand how the assembler does it, let us first see how VARS is structured. With Program Two loaded, add the following lines:

9000 LET Z = PEEK 23627 + 256 x PEEK 23628

9010 FOR I = Z TO Z + 200: PRINT I, PEEK I: NEXT I

Now RUN and, when the maze appears, press BREAK and type GOTO 9000. You will see a screen display of the first 200 bytes of VARS. The first number will be 97. Look at Appendix A of the manual and you will see that this is the character code for "a". This is the first variable to be stored on RUNning. The "0,0,3,0,0" numbers which follow

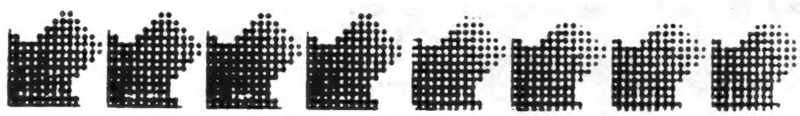



represent the initial value of "a". Keep scrolling and you will see that VARS has been expanded to suit the sequence in which variables are met by the program. Those of interest are for ghost and pacman coordinates, contained in lines 100-120 of the Basic.

**How GCODE works**

Like good Basic programs, good assembler is put together from building blocks called modules. The modules in Figure 1 are numbered according to sequence of development, as it is usual to develop subroutines before the main program.

**Module 1: DATA (lines 40-60).** These three lines set up the first 27 bytes of m/c with the character codes for the variable names. You will see that some of these appear twice. We know that the first part of the MAIN module will deal with storing current ghost coordinates as old, so we will be accessing current, old, current, old, etc., for all eight coordinates. This order is defined in the first two lines of DATA. The second part will update the current ghost coordinates once pacman coordinates have been read, so only current variables appear in the third line.



The DEFB mnemonic means "DEFine Byte". It is NOT a Z80 mnemonic because it is not assembled into operation codes like the mnemonics seen so far. Instead, it is called an "assembler directive", used by most assembler programs. The directive in this case is that the assembler should decode the variable names into character codes and fill out as many bytes with these as required (except for the zero which will be used to mark the end of data).

In essence, DATA tells the CPU in advance what character codes will be scanned in VARS. Ghost and pacman coordinates are accessed frequently and frequently used data is usually specified in this way.

**Module 2: SCAN (lines 840-920).** This m/c subroutine is the real work horse of the routine as a whole, and works by scanning VARS for a particular coordinate. Input data is the character code to be scanned or, rather, a pointer to tell the CPU where it can be found in the DATA. We will use IX register pair for this memory pointer.

The output is also a memory pointer showing where the value of the variable is stored in VARS. Choice of registers other than IX is dictated by the key instruction

**Figure 1**

```
                 10 ;GHOSTCHASE ASSEMBLER
                 20 ;by Tony Rickwood
                 30 ;
616B626C         40 DATA   DEFB "a","k","b","l","c","m","d","n"
656F6670         50        DEFB "e","o","f","p","g","q","h","r"          MODULE 1
78796162         60        DEFB "x","y","a","b","c","d","e","f","g","h",0
                 70 ;
                 80 ;MAIN PROGRAM
                 90 ;
DD2191BE        100        LD   IX,DATA
0608            110        LD   B,8
                120 ;
CD2A8F          130 OLD    CALL SCAN
7E              140        LD   A,(HL)
57              150        LD   D,A
DD23            160        INC  IX
CD2A8F          170        CALL SCAN
72              180        LD   (HL),D
DD23            190        INC  IX
10F1            200        DJNZ OLD
                210 ;
CD2A8F          220 NEW    CALL SCAN
46              230        LD   B,(HL)
DD23            240        INC  IX
CD2A8F          250        CALL SCAN
4E              260        LD   C,(HL)
DD23            270        INC  IX
DD7E00          280 NEXT   LD   A,(IX)
FE00            290        CP   0
C8              295        RET  Z
CD2A8F          296        CALL SCAN
5E              300        LD   E,(HL)
DD23            310        INC  IX
CD2A8F          320        CALL SCAN
56              330        LD   D,(HL)
DD2B            340 XTEST  DEC  IX
78              360        LD   A,B
93              380        SUB  E
281C            390        JR   Z,YTEST
380B            400        JR   C,DECX
1C              410 INCX   INC  E
CD3A8F          420        CALL ATTR                 MODULE 2
FE06            430        CP   6
200E            440        JR   NZ,ENDX
1D              450        DEC  E
180F            460        JR   YTEST
1D              470 DECX   DEC  E
CD3A8F          480        CALL ATTR
FE06            490        CP   6
2003            500        JR   NZ,ENDX
1C              510        INC  E
1804            520        JR   YTEST
CD2A8F          530 ENDX   CALL SCAN
73              540        LD   (HL),E
DD23            550 YTEST  INC  IX
79              570        LD   A,C
92              590        SUB  D
281C            600        JR   Z,END
380B            610        JR   C,DECY
14              620 INCY   INC  D
CD3A8F          630        CALL ATTR
FE06            640        CP   6
200E            650        JR   NZ,ENDY
15              660        DEC  D
180F            670        JR   END
15              680 DECY   DEC  D
CD3A8F          690        CALL ATTR
FE06            700        CP   6
2003            710        JR   NZ,ENDY
14              720        INC  D
1804            730        JR   END
CD2A8F          740 ENDY   CALL SCAN
72              750        LD   (HL),D
DD23            760 END    INC  IX
CD3A8F          770        CALL ATTR
3606            780        LD   (HL),6
18A3            800        JR   NEXT
```

*continued*

```
                820 ;SUBROUTINES
                830 ;
C5              840 SCAN    PUSH BC
2A4B5C          850         LD   HL,(23627)
01C800          860         LD   BC,200
DD7E00          870         LD   A,(IX)
EDB1            880         CPIR               MODULE 3
23              890         INC  HL
23              900         INC  HL
C1              910         POP  BC
C9              920         RET
D5              930 ATTR    PUSH DE
C5              940         PUSH BC
210058          950         LD   HL,22528
4B              960         LD   C,E
5A              970         LD   E,D
1600            980         LD   D,0
0605            990         LD   B,5
CB23           1000 MULT    SLA  E
CB12           1010         RL   D
10FA           1020         DJNZ MULT
7B             1030         LD   A,E           MODULE 4
89             1040         ADD  A,C
3001           1050         JR   NC,HLSET
14             1060         INC  D
5F             1070 HLSET   LD   E,A
19             1080         ADD  HL,DE
7E             1090         LD   A,(HL)
C1             1100         POP  BC
D1             1110         POP  DE
C9             1120         RET
```

at line 880.

CPIR is a block handling instruction read as "ComPare, Increase and Repeat". It searches a number of bytes of memory (specifed in BC) for the first occurrence of a byte (specified in register A). HL is used as the base address (where we want the search to start). It will finish holding the address of the byte immediately following the byte (if found).

Here, the base HL is set to the address pointed to by the system variable VARS (line 850). The first 200 bytes (which we know will hold all our coordinate data) are to be searched (line 860). The byte to be searched out is pointed to by IX (line 870).

After the CPIR, HL will point to the byte after the character code of variable. The coordinate value itself is two bytes on from this so HL must be INCremented twice (lines 890-900).

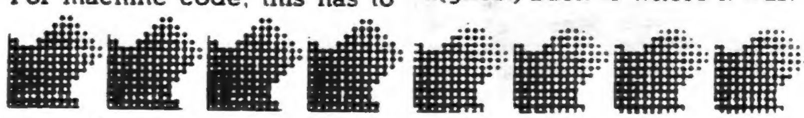
**Module 3: ATTRibute (lines 930-1120).** The output for this subroutine will be the value of the attributes at a new ghost position (which will be tested for collision with a maze wall in the MAIN routine). This result will be stored in register A. Input is the position to be tested and is held in DE (D = y, E = x).

HL is, once again, a memory pointer; this time for the attribute file (line 950). Lines 960-1020 convert the y coordinate to the number of bytes into the attribute file needed to get to the start of the row containing the test position. This means multiplying y by 32 (32 bytes for each row).

For machine code, this has to be thought of as multiplying by two five times, which is done by shifting the bit pattern of the value of y five times to the left. As y x 32 requires two bytes, lines 960-980 get x out of the way for the moment (LD C,E) so that y can go into register E with D=O. Lines 990-1020 are a DJNZ loop to do the multiplication. The rest of the subroutine brings back x and adds the result to the start of the attribute file.

**Module 4: MAIN (lines 80-800).** First, our DATA memory pointer IX is set to the address of the first byte of data. Lines 110-200 (OLD) store all the current ghost coordinates as old, so that old ghosts can be erased on return to Basic.

The next section, called NEW, continues with the main task of moving the ghosts nearer to pacman. Pacman coordinates, x and y are placed in registers B,C (lines 220-260). IX is now incremented ready for "a" (next variable in the DATA list). The new cordinates are updated in the sequence seen in the third line of DATA, with the NEW loop being terminated by testing for end of data (lines 280-295).



The y,x coordinates of each ghost in turn are set up in registers DE (lines 296-330). Lines 340-540 (XTEST) manipulate the x coordinate in E. First, we need to know x(ghost) – x(pacman). The 3 possibilities are contrc 'led as follows:

a) x(pacman) = x(ghost): move to YTEST

b) x(pacman) < x(ghost): move to DECX (decrement x(ghost))

c) x(pacman) > x(ghcs' move to INCX (increment x(ghost))

INCX (lines 410-460): The ATTRibute subroutine is called using the current y(ghost) with the increased x(ghost). CP 6 (line 430) compares the attributes at the new position (held in register from ATTR) with the code for yellow ink on black paper (which indicates a wall).

A non zero result means that the new x(ghost) is valid, so control moves to ENDX (line 440). A zero result means collision with a wall, so line 450 DECrements x(ghost) back to where it was.

Control then moves to YTEST. DECX (lines 470-520) does the same as INX in the opposite sense. Lines 530-540 (ENDX) terminate the XTEST section with a CALL SCAN to get the new x(ghost) entered into VARS (line 540).

The last section, YTEST (lines 550-750) update y(ghost) in the same way as XTEST works on x(ghost).

Finally, END (lines 760-800), increments IX ready for the next ghost variables to be read. More important, it changes the attributes at the new ghost position just calculated to make the ghost look like a wall, at least until it is printed as a ghost on return to Basic. This prevents other ghosts from occupying the same position.